

Response to Office Action
U.S. Patent Appln. No. 09/596,257

Docket No. 6169-155
IBM Docket No. BOC9-2000-0012

REMARKS/ARGUMENTS

The amendments made herein are in response to the Office Action of March 3, 2003 (Office Action). The period for response has been extended for one month to July 3, 2003, by a Petition for an Extension of Time Under 37 C.F.R. §1.136 and a check for the appropriate fee under 37 C.F.R. §1.17 filed concurrently herewith.

Claims 1-2, 4-5, 9-10, and 12-13 have been rejected under 35 U.S.C. § 103(a), as being unpatentable over a Riehle, Dirk [1996] "The Event Notification Pattern: Integrating Implicit Invocation with Object-Orientation" in Theory and Practice of Object Systems, 2, 1, pages 43-52 (Riehle). Claims 3, 6-8, 11, and 14-16 have been rejected under 35 U.S.C. § 103(a), as being unpatentable over Riehle in view of OMG TC Document 95.8.19 [1995] "COM/CORBA Interworking RFP Part A" (OMG) and in further view of Sun Microsystems, Inc. "Remote Method Invocation Specification (Sun RMI). In response, the Applicants have amended claims 4 and 12 to clarify that the Notifier instance and the Listener instance are configured to perform location transparent event handling. No new matter has been added.

Prior to addressing the rejections on the art, a brief review of the Applicant's invention is in order. The Applicants' invention combines event-driven object-oriented programming with distributed object-oriented programming techniques to provide location transparent event handling. The location transparent event handling can be implemented in the Java programming language in a distributed computing environment. Notably, the location transparent event handler can permit event handling between multiple Java Virtual Machines (hereinafter JVMs) which can reside in one or more computer systems and which can stand alone, or be communicatively connected to one another within a network.

The communications can occur through an event notification pattern, wherein an event broadcasting object called a notifier is located within a client JVM and an event listening object called a listener is located within a server JVM. The listening object in the JVM server can be registered with a remote method invocation (RMI) registry. Once

Response to Office Action
U.S. Patent Appln. No. 09/596,257

Docket No. 6169-155
IBM Docket No. BOC9-2000-0012

registered, the listener can include methods that utilize location transparent procedural calls.

Accordingly, the Applicants' invention provides an elegant solution for two previous deficiencies heretofore existing within Java programming models. First, the Java event model provides event handling in a fashion limited to a single virtual machine and is therefore limited to a single computing device. That is, the ability to notify and receive events across virtual machines or across distributed computing devices is not included within the Java event model. Second, a remote method invocation (RMI) model within Java provides a method wherein objects that are written in the Java programming language can interact with other objects within a distributed network. The RMI model lacks event handling capabilities. The Applicants' invention resolves the deficiencies of both models resulting in a generalized solution for adding remote event handling capabilities within the Java programming model.

Turning now to the rejections on the art, claims 1-2, 4-5, 9-10, and 12-13 have been rejected in under 35 U.S.C. §103(a) as being unpatentable over Riehle. Riehle discloses the notion of implicit invocation with object-oriented systems as a decoupling mechanism between objects. That is, Riehle provides a means for detecting a change of state in a software object and responsively updating dependant objects using an event notification pattern. In the pattern, an event broadcaster is disposed within the software object and listeners are disposed within each dependent object. Riehle does not teach placing a notifier object and a listener object in separate address spaces.

Referring to claim 1 and 9, the Examiner has asserted that Riehle discloses using an event notification pattern in a manner that renders the Applicants' invention obvious. The Applicants disagree. The event notification pattern taught by Riehle is specifically configured for implicitly invoking events in order to maintain state relationships among a hierarchy of dependant software objects. Unlike the Applicant's invention, Riehle does not explicitly place listener and notifier objects within separate address spaces as conceded by the Examiner. Because the separation of address spaces is so critical to the Applicants' invention as emphasized throughout the

Response to Office Action
U.S. Patent Appln. No. 09/596,257

Docket No. 6169-155
IBM Docket No. BOC9-2000-0012

specification, the features of claims 1 and 9 are to be interpreted with the listener and notifier objects existing within separate address spaces.

Not only does Riehle fail to disclose the use of different address spaces, but the very nature of Riehle does not lend itself to their use. Riehle discloses a method for maintaining state relationships between interdependent software objects that presupposes that a close, intermeshed relationship exists between the software objects. When such a relationship exists, it can be assumed that the interdependent software objects operate in an tightly integrated environment characteristic of an environment having a single address space, as opposed to a segmented environment with several distinct address spaces. Dependencies between objects would not typically need to be maintained across a loosely coupled amalgamation of objects existing within separate address spaces. In fact, Riehle does not disclose a situation where interdependent objects would be located in separate address spaces in either the cited document or within the numerous code examples included within Riehle.

The Examiner asserts, however, that Riehle teaches that the event notification pattern can be used in an distributed system citing page 1. The citation refers to the event notification pattern in general and not the pattern as specifically applied to the method of Riehle. Riehle explicitly contrasts its teachings with other event notification teachings by stating on page 1 that "[t]his paper focuses on the decoupling of object interfaces." This distinction between method taught by Riehle and a generic event notification pattern is elaborated upon at page 2, paragraph 10, where Riehle states that the paper "concentrates on the problems of maintaining inter-object update dependencies" as opposed to other ways of using event notification methods, such as use within distributed systems cited immediately above. The Applicants note that the assertion that "distributed systems are well supported for big projects" is unclear and clarification is respectfully requested to the extent the assertion is relevant in light of the remarks included herein.

Not only does Riehle not disclose placing a notifier within a client and a listener in a server, but Riehle in fact teaches away from such a placement. If one were to place interdependent objects in a distributed environment, an independent object (that

Response to Office Action
U.S. Patent Appl. No. 09/596,257

Docket No. 6169-155
IBM Docket No. BOC9-2000-0012

cascades changes to dependent objects) should be disposed within a server and dependent objects should be disposed within a client. Examiner suggests, however, placing the independent object within a client (since the independent object contains the notifier) and the dependent objects within a server (since the dependent objects contain the listener). Appreciably, users of a client computing device often have lower access privileges than users of server computing devices. Accordingly, users with server privileges can typically modify information of client applications, yet users with client privileges cannot often modify information of server applications due to general principles of system security and maintenance. The suggested arrangement for Riehle, however, violates this basic computing principle by requiring an arrangement where client behavior automatically results in server changes.

Further, by nature, changes within a server can affect a multitude of client applications. One of the intended purposes of Riehle, as noted in the abstract and at page 2, paragraph 4, is to prevent cyclic dependencies that hamper system evolution and maintenance by intertwining objects. That is, Riehle discloses the placing of the notifier in the independent object and the listener in dependant objects so that changes in dependent objects do not cause changes in the independent object potentially resulting in further changes in the dependent objects. Placing a notifier object within a server and a listener in a client can result in the type of cyclic behavior that Riehle is designed to prevent because client programs are naturally affected by changes that occur within a corresponding server program.

Referring to claims 2, 5, 10, and 13 the Examiner has conceded that Riehle does not disclose the placement of a notifier and a listener within different JVMs. The Examiner asserts, however, that it would have been obvious based upon Riehle to implement the notifier and listener classes within different JVMs. The Applicants disagree as Riehle maintains inter-object dependencies that would not exist across different JVMs. That is, each JVM by definition functions as a separate, autonomous computing environment. No implicit parameter passing/ memory sharing/ or process sharing occurs between two different JVMs. Consequently, objects that are

Response to Office Action
U.S. Patent Appln. No. 09/596,257.

Docket No. 6169-155
IBM Docket No. BOC9-2000-0012

interdependent upon one another would not be placed within different JVMs. Therefore, one would not place listeners and notifiers within different JVMs based upon Riehle.

Referring to claims 4 and 10, the Examiner asserts that placing a notifier and a listener in separate address spaces is taught or suggested by Riehle. The Applicants reassert that Riehle does not disclose a method that calls remotely located procedures in response to a detected event. Further, Riehle does not disclose a method or apparatus where the Notifier instance and the Listener instance are configured to perform location transparent event handling as currently claimed.

In paragraph 5, claims 3, 6, 11, and 14 have been rejected in under 35 U.S.C. §103(a) as being unpatentable over Riehle in view of OMG and further in view of Sun RMI. OMG discloses interfaces and an architecture known as the Common Object Request Broker Architecture (CORBA). CORBA is an architecture and specification for creating, distributing, and managing distributed program objects in a network. It allows programs at different locations and developed by different vendors to communicate in a network through an "interface broker." In the CORBA architecture, CORBA objects that are linked to interfaces can be remotely accessed by other CORBA objects. In contrast to the Applicants' claimed invention, the CORBA architecture can only establish communications between CORBA objects and Object Linking and Embedding(OLE) objects.

Teachings in OMG should not be combined with Riehle since OMG teaches away from Riehle. The event handling disclosed by OMG at page 4, paragraph 2, however, states that events can be managed by mapping OLE Custom Controls (OCX's) to propagate events via OCX interfaces. That is, OLE objects in CORBA require a coupling between communication objects and an interface. Further, according to the CORBA specification, only CORBA objects with interfaces can be accessed remotely. Riehle, however, discloses at page 1, paragraph 5, that interdependent software objects should not be coupled to interfaces or interface objects so that system evolution and maintenance is facilitated.

Turning to claims 7-8 and 15-16, the Examiner has attempted to cure the deficiencies of Riehle and OMG by referencing Sun RMI. Sun RMI discloses an

Response to Office Action
U.S. Patent Appln. No. 09/596,257

Docket No. 6169-155
IBM Docket No. BOC9-2000-0012

architecture through which a programmer using Java can write object-oriented programming in which objects on different computers can interact in a distributed network. The RMI model fails to include event handling procedures. According to the RMI specification, methods calling an RMI registered method explicitly invoke the method by name. Thereafter, an associated resource, that can be remotely located, is triggered. In contrast, the Applicant's invention detects an event which implicitly and automatically invokes a RMI registered method. No explicit invocation occurs.

Notably, RMI does not include event handling routines even though a competing technology, CORBA, does include event handling capabilities. An event handling model, however, is included within Java that is not capable of being utilized across different JVMs, which is a requirement for performing error handling in conjunction with RMI. The lack of event handling within RMI therefore implies that either (1) Sun Microsystems has failed to recognize the need for remote event handling across different JVMs and/or that (2) Sun Microsystems has been unable to successfully implement remote event handling across different JVMs. In either case, the lack of an event handling capabilities within an often utilized and mature RMI model is a testament to the innovative nature of the Applicants' invention.

It should be recognized that RMI and CORBA are different, competing technologies for establishing communications between distributed objects. A variety of structural and functional differences exist between the RMI and CORBA, resulting in the technologies being incompatible with one another. Examples of some significant deviations between RMI and CORBA include:

- RMI interfaces are defined in Java, while CORBA interfaces are defined in Interface Definition Language (IDL).
- RMI relies heavily upon Java Object Serialization to function and can be run upon any platform that implements a JVM. In contrast, CORBA relies upon ORB libraries to be written that adhere to the CORBA specification to function and are not natively operable within a JVM.
- RMI passes local objects by copying them and passes remote objects by reference allowing new classes to be passed across JVM for execution (mobile code). In contrast, CORBA supports passing parameters according to input and

Response to Office Action
U.S. Patent Appln. No. 09/596,257

Docket No. 6169-155
IBM Docket No. BOC9-2000-0012

output definitions allowing only primitive data types and structures to be passed, as opposed to actual code.

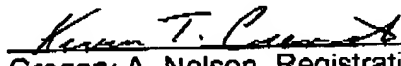
- RMI automatically performs garbage collecting on distributed objects using mechanisms located within a JVM, while CORBA objects garbage collection is not performed upon CORBA objects.
- RMI is a model designed for a single language where all objects are written in Java. CORBA is a language independent specification.

The RMI and CORBA technologies have different advantages and disadvantages over one another. Event handling concepts that function within CORBA cannot be simply integrated into the competing RMI model as suggested by the Examiner. In fact, porting CORBA event handling concepts so that such functions could operate within an RMI model would require substantial innovations not known within the relevant art at the time of the invention.

In light of the above discussion, neither Riehle, OMG, Sun RMI, nor any combination thereof teach or suggest the Applicant's invention as claimed. Therefore, withdrawal of the 35 U.S.C. § 103(a) rejection of claims 1-16 is respectfully requested. The Applicants believe that this application is now in full condition for allowance, which action is respectfully requested. The Applicants request that the Examiner call the undersigned if clarification is needed on any matter within this Amendment, or if the Examiner believes a telephone interview would expedite the prosecution of the subject application to completion.

Respectfully submitted,

Date: 7/3/03


Gregory A. Nelson, Registration No. 30,577
Kevin T. Cuenot, Registration No. 46,283
Brian K. Buchheit, Registration No. 52,667
AKERMAN SENTERFITT
222 Lakeview Avenue, Suite 400
Post Office Box 3188
West Palm Beach, FL 33402-3188
Telephone: (561) 653-5000